Projeto *Kiosk-IUL* (Parte 3)

O presente trabalho visa aplicar os conhecimentos adquiridos durante as aulas de Sistemas Operativos e será composto por três partes, com o objetivo de desenvolver os diferentes aspetos da plataforma **Kiosk-IUL**. Iremos procurar minimizar as interdependências entre partes do trabalho.

Este enunciado detalha apenas as funcionalidades que devem ser implementadas na parte 3 do trabalho.

A aplicação **Kiosk-IUL** permite gerir e fazer compras num quiosque de mercearias no campus do ISCTE que está aberto 24h e é totalmente automatizado. Na aplicação **Kiosk-IUL** existem os seguintes conceitos:

Algumas definições básicas e tipos de dados para interoperabilidade entre **Cliente** e **Servidor** estão no ficheiro **common.h**:, sendo a estrutura `Login` igual à da parte 2 do trabalho, e acrescentam-se as seguintes:

```
typedef struct {
    int idProduto;                  // Identificador único do produto
    char nomeProduto[40];           // Nome do Produto
    char categoria[40];             // Categoria do Produto
    int preco;                      // Preço do Produto
    int stock;                      // Stock do Produto
} Produto;

typedef struct {
    long msgType;                   // Tipo da Mensagem
    struct {
        Login infoLogin;            // Informação sobre o Login
        Produto infoProduto;        // Informação sobre um Produto
    } msgData;                      // Dados da Mensagem
} MsgContent;
```

Os alunos deverão, em vez de **printf** (não será analisado para efeito de avaliação), utilizar sempre as macros **`so_success`** (para as mensagens de sucesso) e **`so_error`** (para as mensagens de erro) definidas no *header file* **/home/so/reference/so_utils.h** (cuja sintaxe está descrita na KB C Language & Compiling / Para que serve o header file "/home/so/reference/so_utils.h?"), indicando SEMPRE a alínea correspondente (e.g., **`so_error("S2", "");`**), e sempre que no enunciado estiverem indicados os pedidos de valores entre **< >**, o aluno deverá (naturalmente!) substituir esse texto pelos valores indicados (e.g., **`so_success("S4", "%d", pid_servidor);`**), garantindo que é sempre cumprida estritamente a especificação apresentada sem acrescentar mais informação.

A *baseline* para o trabalho encontra-se no Tigre, na diretoria **`/home/so/trabalho-2022-2023/parte-3`**. É obrigatório que os alunos trabalhem com base **nestes** ficheiros e não com base em ficheiros vazios.

- Para tal, **EXECUTE**, **a partir da sua diretoria local de projeto**, os seguintes comandos:

```
$ cp -r /home/so/trabalho-2022-2023/parte-3 .
```

Resultado:

```
bd_produtos.dat
bd_utilizadores.dat
cliente.c
common.h -> utils/common.h
servidor.c
so-2022-trab3-validator/
so_utils.h -> /home/so/reference/so_utils.h
utils -> /home/so/trabalho-2022-2023/utils/parte-3/
```

Procedimento de entrega e submissão do trabalho

O trabalho de SO será realizado **individualmente**, logo sem recurso a grupos.

A entrega da Parte 3 do trabalho será realizada através da criação de **um** ficheiro ZIP cujo nome é o nº do aluno, e.g., "**a<nºaluno>-parte-3.zip**" (**ATENÇÃO:** não serão aceites ficheiros RAR, 7Z ou outro formato) onde estarão todos os ficheiros criados. Estes serão **apenas** os ficheiros de código, ou seja, na parte 3, apenas os ficheiros (***.c *.h**).

Cada um dos módulos será desenvolvido com base nos ficheiros fornecidos, e que estão na diretoria do Tigre **"/home/so/trabalho-2022-2023/parte-3"**, e deverá incluir nos comentários iniciais um "relatório" indicando a descrição do módulo e explicação do mesmo (poderá ser muito reduzida se o código tiver comentários bem descritivos).

Para criarem o ficheiro ZIP para submissão do trabalho, posicionem-se no Tigre na diretoria **parte-3**, e executem:

```
$ zip $USER-parte-3.zip *.c *.h
```

O ficheiro ZIP deverá depois ser transferido do Tigre para a vossa área local (Windows/Linux/Mac) via SFTP, para depois ser submetido via Moodle (ver no Moodle a KB Basics / Criar ficheiro ZIP para submeter trabalho no Moodle).

Antes de submeter, por favor validem que o ficheiro ZIP não inclui diretorias ou ficheiros extra indesejados.

A entrega desta parte do trabalho deverá ser feita por via eletrónica, através do Moodle:

- Moodle da UC Sistemas Operativos, Selecionam a opção sub-menu "Quizzes & Assignments";
- Selecionem o link "Submit SO Assignment 2022-2023 Part 3";
- Dentro do formulário, selecionem o botão "Enviar trabalho" e anexem o vosso ficheiro .zip (a forma mais fácil é simplesmente fazer via *drag-and-drop*) e selecionar o botão "Guardar alterações". Podem depois mais tarde ressubmeter o vosso trabalho as vezes que desejarem, enquanto estiverem dentro do prazo para entrega do trabalho. Para isso, na mesma opção, pressionar o botão "Editar submissão", selecionar o ficheiro, e depois o botão "Apagar", sendo que depois pode arrastar o novo ficheiro e pressionar "Guardar alterações". **Apenas a última submissão será contabilizada**. Certifiquem-se que a submissão foi concluída, e que esta última versão tem todas as alterações que desejam entregar dado que os docentes apenas considerarão esta última submissão;
- Avisamos que a hora *deadline* acontece sempre poucos **minutos antes da meia-noite**, pelo que se urge a que os alunos não esperem por essa hora final para entregar e o façam antes, idealmente um dia antes, ou no pior dos casos, pelo menos uma hora antes. **Não serão consideradas válidas as entregas com ficheiros com nomes diferentes do especificado acima, nem entregas realizadas por e-mail**. Poderão testar a entrega nos dias anteriores para perceber se há algum problema com a entrega, sendo que, **apenas a última submissão conta**.

Política em caso de fraude

O trabalho corresponde ao esforço individual de cada aluno. São consideradas fraudes as seguintes situações: Trabalho parcialmente copiado, facilitar a cópia através da partilha de ficheiros, ou utilizar material alheio sem referir a fonte.

Em caso de deteção de fraude, os trabalhos em questão não serão avaliados, sendo enviados à Comissão Pedagógica da escola (ISTA) ou ao Conselho Pedagógico do ISCTE, consoante a gravidade da situação, que decidirão a sanção a aplicar aos alunos envolvidos. Serão utilizadas as ferramentas *Moss* e *SafeAssign* para deteção automática de cópias.

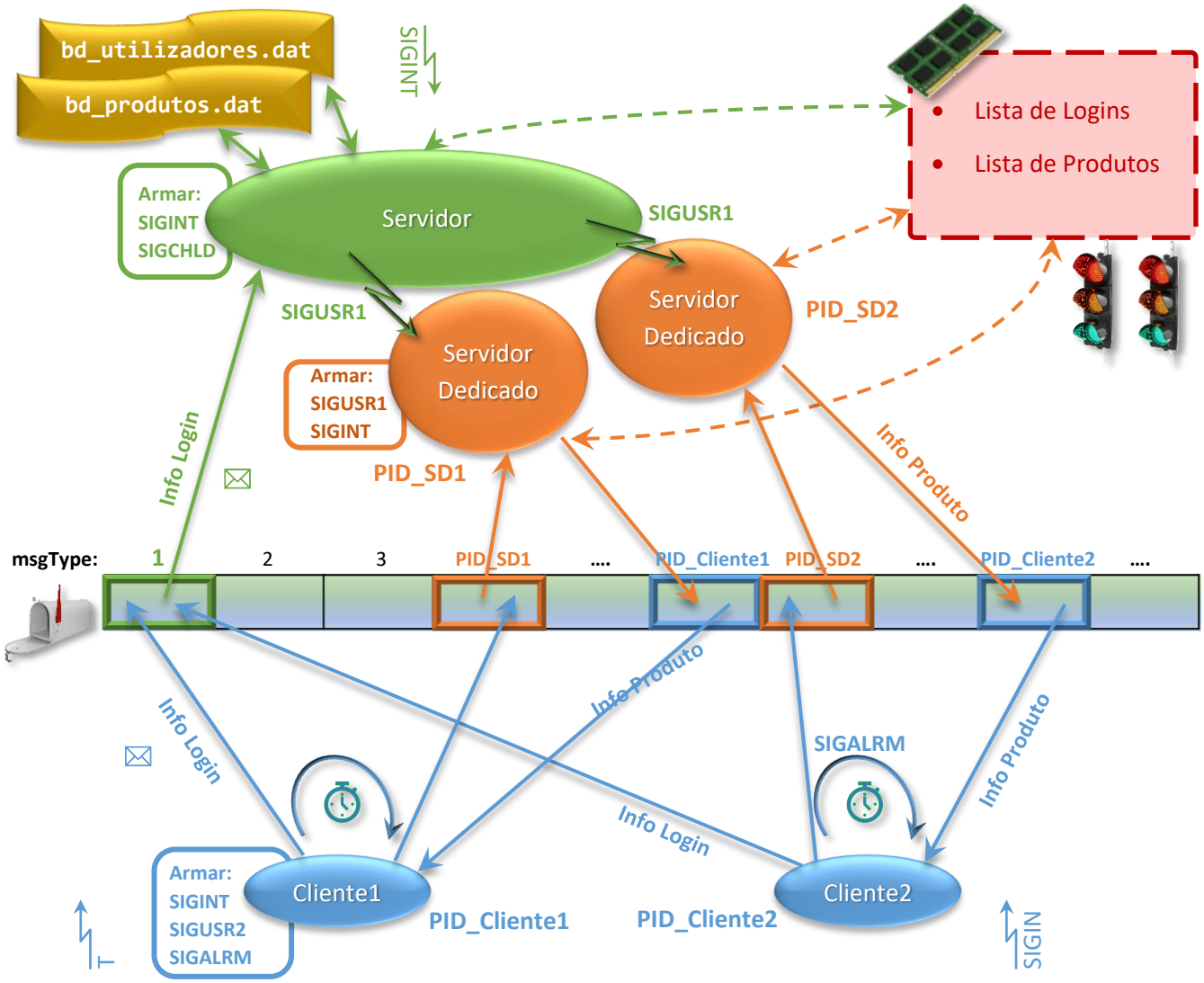
Recorda-se ainda que o Anexo I do Código de Conduta Académica, publicado a 25 de janeiro de 2016 em Diário da República, 2ª Série, nº 16, indica no seu ponto 2 que quando um trabalho ou outro elemento de avaliação apresentar um nível de coincidência elevado com outros trabalhos (percentagem de coincidência com outras fontes reportada no relatório que o referido software produz), cabe ao docente da UC, orientador ou a qualquer elemento do júri, após a análise qualitativa desse relatório, e em caso de se confirmar a suspeita de plágio, desencadear o respetivo procedimento disciplinar, de acordo com o Regulamento Disciplinar de Discentes do ISCTE - Instituto Universitário de Lisboa, aprovado pela deliberação nº 2246/2010, de 6 de dezembro.

O ponto 2.1 desse mesmo anexo indica ainda que no âmbito do Regulamento Disciplinar de Discentes do ISCTE-IUL, são definidas as sanções disciplinares aplicáveis e os seus efeitos, podendo estas variar entre a advertência e a interdição da frequência de atividades escolares no ISCTE-IUL até cinco anos.

Parte 3 – Processos e Sinais

Data de entrega: 21 de maio de 2023

Nesta parte do trabalho, será implementado um modelo simplificado de gestão da autenticação de um quiosque de compras automáticas, **Kiosk-IUL**, baseado em comunicação por sinais entre processos, utilizando a linguagem de programação C. Considere o seguinte diagrama, que apresenta uma visão geral da arquitetura pretendida:



Pretende-se, nesta fase, simular um servidor de sessões de autenticação de utilizadores no quiosque **Kiosk-IUL**. Assim, teremos dois módulos – **Cliente** e **Servidor**. Os conhecimentos que se pretende que os alunos sejam avaliados com este trabalho são:

- Utilização de memórias partilhadas IPC (**shmget**, **shmat**, **shmdt**);
- Concorrência entre processos usando semáforos IPC (**semget**, **semctl**, **semop**);
- Comunicação usando filas de mensagem IPC (**msgget**, **msgsnd**, **msgrcv**).

Atenção: Apesar do vários ficheiros necessários para a realização do trabalho serem fornecidos na diretoria do Tigre "**/home/so/trabalho-2022-2023/parte-3**", assume-se que, para a sua execução, os scripts e todos os ficheiros de input e de output estarão todos **sempre** presentes na mesma diretoria, que não deve estar *hard-coded*, ou seja, os programas entregues devem correr em qualquer diretoria.

1. Módulo: `servidor.c`

O módulo **Servidor** é responsável pelo processamento das autenticações dos utilizadores. Está dividido em duas partes, o **Servidor** (pai) e zero ou mais **Servidores Dedicados** (filhos).

ATENÇÃO: Lembra-se que estes processos atuam de forma concorrente, acedendo a recursos partilhados, pelo que os alunos deverão cuidar para que sejam **definidos e utilizados os mecanismos de exclusão mútua no acesso aos mesmos**, sendo que essa exclusão deverá ser feita pelo menor tempo possível (dentro dos limites razoáveis, claro), e sem nunca permitir que haja esperas ativas. Este módulo realiza as seguintes tarefas:

S1 Abre/Cria a *Shared Memory* (SHM) do projeto, que tem a KEY **IPC_KEY** definida em **common.h** (alterar esta KEY para ter o valor do nº do aluno, como indicado nas aulas), realizando as seguintes operações:

- **S1.1** Tenta abrir a *Shared Memory* (SHM) IPC com a referida KEY **IPC_KEY**. Em caso de sucesso na abertura da SHM, liga a variável db a essa SHM. Se não encontrar nenhum erro, dá **`so_success <shmId>`** (i.e., **`so_success("S1.1", "%d", shmId)`**), e retorna o ID da SHM (vai para **S2**). Caso contrário, dá **`so_error`**, (i.e., **`so_error("S1.1", "")`**).
- **S1.2** Valida se o erro anterior foi devido a não existir ainda nenhuma SHM criada com essa KEY (testando o valor da variável `errno`). Se o problema não foi esse (mas outro qualquer), então dá **`so_error`** e retorna erro (vai para **S7**). Caso contrário (o problema foi não haver SHM criada), dá **`so_success`**.
- **S1.3** Cria uma *Shared Memory* com a KEY **IPC_KEY** definida em **common.h** e com o tamanho para conter as duas listas do **Servidor**: uma de utilizadores (**Login**), que comporta um máximo de **MAX_USERS** elementos, e outra de produtos (**Produto**), que comporta um máximo de **MAX_PRODUCTS** elementos, e liga a variável db a essa SHM. Em caso de erro, dá **`so_error`** e retorna erro (vai para **S7**). Caso contrário, dá **`so_success <shmId>`**.
- **S1.4** Inicia a Lista de Utilizadores, preenchendo em todos os elementos o campo **`nif`=USER_NOT_FOUND** ("Limpa" a lista de utilizadores), e a Lista de Produtos, preenchendo em todos os elementos o campo **`idProduto`=PRODUCT_NOT_FOUND** ("Limpa" a lista de produtos). No final, dá **`so_success`**.
- **S1.5** Lê o ficheiro **`bd_utilizadores.dat`** e preenche a lista de utilizadores na memória partilhada com a informação dos utilizadores, mas preenchendo sempre os campos pidCliente e pidServidorDedicado com o valor **`-1`**. Em caso de qualquer erro, dá **`so_error`** e retorna erro (vai para **S7**). Caso contrário, dá **`so_success`**.
- **S1.6** Lê o ficheiro **`bd_produtos.dat`** e preenche a lista de produtos na memória partilhada com a informação dos produtos. Em caso de qualquer erro, dá **`so_error`** e retorna erro (vai para **S7**). Caso contrário, dá **`so_success`** e retorna o ID da SHM.

S2 Cria a *Message Queue* (MSG) do projeto, que tem a KEY **IPC_KEY**, realizando as seguintes operações:

- **S2.1** Se já existir, deve apagar a fila de mensagens. Em caso de qualquer erro, dá **`so_error`** e retorna erro (vai para **S7**). Caso contrário, dá **`so_success`**.
- **S2.2** Cria a *Message Queue* com a KEY **IPC_KEY**. Em caso de erro, dá **`so_error`** e retorna erro (vai para **S7**). Caso contrário, dá **`so_success <msgId>`** e retorna o ID da MSG.

S3 Cria um grupo de semáforos (SEM) que tem a KEY **IPC_KEY**, realizando as seguintes operações:

- **S3.1** Se já existir, deve apagar o grupo de semáforos. Em caso de qualquer erro, dá **`so_error`** e retorna erro (vai para **S7**). Caso contrário, dá **`so_success`**.
- **S3.2** Cria um grupo de três semáforos com a KEY **IPC_KEY**. Em caso de qualquer erro, dá **`so_error`** e retorna erro (vai para **S7**). Caso contrário, dá **`so_success <semId>`**.

S3.3 Inicia o valor dos semáforos **SEM_USERS** e **SEM_PRODUCTS** para que possam trabalhar em modo "exclusão mútua", e inicia o valor do semáforo **SEM_NR_SRV_DEDICADOS** com o valor `0`. Em caso de erro, dá **`so_error`** e retorna erro (vai para **S7**). Caso contrário, dá **`so_success`** e retorna o ID do SEM.

S4 Arma e trata os sinais **SIGINT** (ver **S8**) e **SIGCHLD** (ver **S9**). Em caso de qualquer erro a armar os sinais, dá **`so_error`** e retorna erro (vai para **S7**). Caso contrário, dá **`so_success`** e retorna sucesso.

S5 Lê da *Message Queue* um pedido, ou seja, uma mensagem do tipo **MSGTYPE_LOGIN**. Se houver erro, dá **`so_error`** e retorna erro (reinicia o processo neste mesmo passo **S5**), lendo um novo pedido. Caso contrário, dá **`so_success <nif> <senha> <pidCliente>`**.

S6 Cria um processo filho (**fork**) **Servidor Dedicado**. Se houver erro, dá **`so_error`** e retorna erro (vai para **S7**). Caso contrário, o processo **Servidor Dedicado** (filho) continua no passo **SD10**, enquanto o processo **Servidor** (pai) dá **`so_success "Servidor Dedicado: PID <pidServidorDedicado>"`**, e retorna o PID do novo processo **`Servidor Dedicado`** (recomeça no passo **S5**).

S7 Passo terminal para fechar o **Servidor**: dá **`so_success "Shutdown Servidor"`**, e faz as seguintes ações:

S7.1 Verifica se existe SHM aberta e alocada. Se não existir, dá **`so_error`** e passa para o passo **S7.5**, caso contrário, dá **`so_success`**.

S7.2 Percorre a lista de utilizadores. A cada utilizador que tenha um **Servidor Dedicado** associado (significando que está num processo de compra de produtos), envia ao PID desse **Servidor Dedicado** o sinal **SIGUSR1**. Quando tiver processado todos os utilizadores existentes, dá **`so_success`**.

S7.3 Dá **`so_success`**.

S7.4 Reescreve o ficheiro **bd_produtos.dat**, mas incluindo apenas os produtos existentes na lista de produtos que tenham stock > `0`. Em caso de qualquer erro, dá **`so_error`**. Caso contrário, dá **`so_success`**.

S7.5 Apaga todos os elementos IPC (SHM, SEM e MSG) que tenham sido criados pelo **Servidor** com a KEY **IPC_KEY**. Em caso de qualquer erro, dá **`so_error`**. Caso contrário, dá **`so_success`**.

S7.6 Termina o processo **Servidor**.

S8 O sinal armado **SIGINT** serve para o dono da loja encerrar o **Servidor**, usando o atalho **<CTRL+C>**. Se receber esse sinal (do utilizador via Shell), o **Servidor** dá **`so_success`**, e vai para o passo terminal **S7**.

S9 O sinal armado **SIGCHLD** serve para que o **Servidor** seja alertado quando um dos seus filhos **Servidor Dedicado** terminar. Se o **Servidor** receber esse sinal, identifica o PID do **Servidor Dedicado** que terminou (usando **wait**), dá **`so_success "Terminou Servidor Dedicado <pidServidorDedicado>"`**, retornando ao que estava a fazer anteriormente.

SD10 O novo processo **Servidor Dedicado** (filho) arma os sinais **SIGUSR1** (ver **SD18**) e **SIGINT** (programa-o para ignorar este sinal). Em caso de erro a armar os sinais, dá **`so_error`** e retorna erro (vai para **SD17**). Caso contrário, dá **`so_success`**.

SD11 O **Servidor Dedicado** deve validar, em primeiro lugar, no pedido **Login** recebido do **Cliente** (herdado do processo **Servidor** pai), se o campo **pidCliente** > `0`. Se for, dá **`so_success`** e retorna sucesso. Caso contrário, dá **`so_error`** e retorna erro (vai para **SD17**).

SD12 Percorre a lista de utilizadores, atualizando a variável `indexClient`, procurando pelo utilizador com o NIF recebido no pedido do **Cliente**.

SD12.1 Se encontrou um utilizador com o NIF recebido, e a Senha registada é igual à que foi recebida no pedido do **Cliente**, então dá **`so_success <indexClient>`**, e retorna indexClient (vai para **SD13**). Caso contrário, dá **`so_error`**.

SD12.2 Cria uma resposta indicando erro ao **Cliente**, preenchendo na estrutura **Login** o campo **pidServidorDedicado**=-1. Envia essa mensagem para a fila de mensagens, usando como msgType o

PID do processo **Cliente**. Em caso de erro, dá **`so_error`**, caso contrário dá **`so_success`**. Em ambos os casos, retorna erro (**USER_NOT_FOUND**, vai para **SD17**).

SD13 Reserva a entrada do utilizador na BD, atualizando na Lista de Utilizadores, na posição `indexClient`, os campos **pidServidorDedicado** e **pidCliente** (com o valor do pedido do **Cliente**), e dá **`so_success`**.

SD14 Cria a resposta indicando sucesso ao **Cliente**:

SD14.1 Preenche na mensagem de resposta os campos **nome** e **saldo** da estrutura **Login** com os valores da Lista de Utilizadores para `indexClient`. Preenche o campo **pidServidorDedicado** com o PID do processo **Servidor Dedicado**, e dá **`so_success`**.

SD14.2 Envia a lista de produtos ao **Cliente**: Percorre a Lista de Produtos, e por cada produto com stock > `0`, preenche a estrutura Produto da mensagem de resposta com os dados do produto em questão, e envia, usando como msgType o PID do processo **Cliente**, a mensagem de resposta ao **Cliente**. Em caso de erro, dá **`so_error`**, e retorna erro (vai para **SD17**). No fim de enviar a lista, dá **`so_success`**.

SD14.3 Depois de ter enviado todas as mensagens (uma por cada produto com stock > `0`), preenche uma nova mensagem final, preenchendo a estrutura Produto novamente, colocando apenas o campo **idProduto**=**FIM_LISTA_PRODUTOS**, o que se convencionou que significa que não há mais produtos a listar. Envia, usando como msgType o PID do processo **Cliente**, a mensagem de resposta ao **Cliente**. Em caso de erro, dá **`so_error`**, e retorna erro (vai para **SD17**). Caso contrário, dá **`so_success`**.

SD15 Lê da fila de mensagens a resposta do **Cliente**: uma única mensagem com msgType igual ao PID deste processo **Servidor Dedicado**, indicando no campo **idProduto** qual foi o produto escolhido pelo **Cliente**. Em caso de erro, dá **`so_error`**, e retorna erro (vai para **SD17**). Caso contrário, dá **`so_success`**.

SD16 Produz a resposta final a dar ao **Cliente**:

SD16.1 Se o **idProduto** enviado pelo **Cliente** for **PRODUCT_NOT_FOUND**, então preenche o campo **idProduto**=**PRODUTO_NAO_COMPRADO** e dá **`so_error`**.

SD16.2 Caso contrário, percorre a lista de produtos, procurando pelo produto com o **idProduto** recebido no pedido do **Cliente**. Se não encontrou nenhum produto com o **idProduto** recebido, ou se encontrou esse produto, mas o mesmo já não tem *stock* (porque, entretanto, já esgotou), preenche o campo **idProduto**=**PRODUTO_NAO_COMPRADO** e dá **`so_error`**. Caso contrário, decrementa o stock do produto na Lista de Produtos, preenche o campo **idProduto**=**PRODUTO_COMPRADO**, e dá **`so_success`**. Atenção: Deve cuidar para que o acesso ao stock do produto seja feito em exclusão!

SD16.3 Envia, usando como msgType o PID do processo **Cliente**, a mensagem de resposta de conclusão ao **Cliente**. Em caso de erro, dá **`so_error`**. Caso contrário, dá **`so_success`**.

SD17 Passo terminal para fechar o **Servidor Dedicado**: dá **`so_success "Shutdown Servidor Dedicado"`**, e, de seguida, faz as seguintes ações:

SD17.1 Atualiza, na Lista de utilizadores para a posição `indexClient`, os campos **pidServidorDedicado**=-1 e **pidCliente**=-1, e dá **`so_success`**.

SD17.2 Termina o processo **Servidor Dedicado**.

SD18 O sinal armado **SIGUSR1** serve para que o **Servidor Dedicado** seja alertado quando o **Servidor** principal quer terminar. Se o **Servidor Dedicado** receber esse sinal, envia um sinal **SIGUSR2** ao **Cliente** (para avisá-lo do Shutdown), dá **`so_success`**, e vai para o passo terminal **SD17**.

2. Módulo: `cliente.c`

O módulo **Cliente** é responsável pela interação com o utilizador. Após o login do utilizador, este poderá realizar atividades durante o tempo da sessão. Assim, definem-se as seguintes tarefas a desenvolver:

C1 Abre a *Message Queue* (MSG) do projeto, que tem a KEY **IPC_KEY**. Deve assumir que a fila de mensagens já foi criada pelo **Servidor**. Em caso de erro, dá **`so_error`** e termina o **Cliente**. Caso contrário dá **`so_success <msgId>`**.

C2 Arma e trata os sinais **SIGUSR2** (ver **C10**), **SIGINT** (ver **C11**), e **SIGALRM** (ver **C12**). Em caso de qualquer erro a armar os sinais, dá **`so_error`** e termina o **Cliente**. Caso contrário, dá **`so_success`**.

C3 Pede ao utilizador que preencha os dados referentes à sua autenticação (**NIF** e **Senha**), criando um elemento do tipo **Login** com essas informações, e preenchendo também o campo **pidCliente** com o PID do seu próprio processo **Cliente**. Os restantes campos da estrutura **Login** não precisam ser preenchidos. Em caso de qualquer erro, dá **`so_error`** e termina o **Cliente**. Caso contrário dá **`so_success <nif> <senha> <pidCliente>`**.

C4 Envia uma mensagem do tipo **MSGTYPE_LOGIN** para a MSG com a informação recolhida do utilizador. Em caso de erro, dá **`so_error`**. Caso contrário, dá **`so_success`**.

C5 Configura um alarme com o valor de MAX_ESPERA segundos (ver **C12**), e dá **`so_success "Espera resposta em <MAX_ESPERA> segundos"`**.

C6 Lê da *Message Queue* uma mensagem cujo tipo é o PID deste processo **Cliente**. Se houver erro, dá **`so_error`** e termina o **Cliente**. Caso contrário, dá **`so_success <nome> <saldo> <pidServidorDedicado>`**.

- **C6.1** "Desliga" o alarme configurado em **C5**.
- **C6.2** Valida se o resultado da autenticação do **Servidor Dedicado** foi sucesso (convenciona-se que se a autenticação não tiver sucesso, o campo **pidServidorDedicado**==-1). Nesse caso, dá **`so_error`**, e termina o **Cliente**. Senão, escreve no STDOUT a frase "Lista de Produtos Disponíveis:".
- **C6.3** Extrai da mensagem recebida o Produto especificado. Se o campo **idProduto** tiver o valor **FIM_LISTA_PRODUTOS**, convencionou-se que significa que não há mais produtos a listar, então dá **`so_success`** e retorna sucesso (vai para **C7**).
- **C6.4** Mostra no STDOUT uma linha de texto com a indicação de idProduto, Nome, Categoria e Preço.
- **C6.5** Lê da *Message Queue* uma nova mensagem cujo tipo é o PID deste processo **Cliente**. Se houver erro, dá **`so_error`** e termina o **Cliente**. Caso contrário, volta ao passo **C6.3**.

C7 Pede ao utilizador que indique qual o **idProduto** (número) que deseja adquirir. Não necessita validar se o valor inserido faz parte da lista apresentada. Em caso de qualquer erro, dá **`so_error`** e retorna **PRODUCT_NOT_FOUND**. Caso contrário dá **`so_success <idProduto>`**, e retorna esse idProduto.

C8 Envia uma mensagem cujo tipo é o PID do **Servidor Dedicado** para a MSG com a informação do **idProduto** recolhida do utilizador. Em caso de erro, dá **`so_error`**. Caso contrário, dá **`so_success`**.

C9 Lê da MSG uma mensagem cujo tipo é o PID deste processo **Cliente**, com a resposta final do **Servidor Dedicado**. Em caso de erro, dá **`so_error`**. Caso contrário, dá **`so_success <idProduto>`**. Se o campo idProduto for **PRODUTO_COMPRADO**, escreve no STDOUT "Pode levantar o seu produto". Caso contrário, escreve no STDOUT "Ocorreu um problema na sua compra. Tente novamente". Em ambos casos, termina o **Cliente**.

C10 O sinal armado **SIGUSR2** serve para o **Servidor Dedicado** indicar que o servidor está em modo *shutdown*. Se o **Cliente** receber esse sinal, dá **`so_success`** e termina o **Cliente**.

C11 O sinal armado **SIGINT** serve para que o utilizador possa cancelar o pedido do lado do **Cliente**, usando o atalho **<CTRL+C>**. Se receber esse sinal (do utilizador via Shell), o **Cliente** dá **`so_success "Shutdown Cliente"`**, e termina o **Cliente**.

C12 O sinal armado **SIGALRM** serve para que, se o **Cliente** em **C6** esperou mais do que MAX_ESPERA segundos sem resposta, o **Cliente** dá **`so_error "Timeout Cliente"`**, e termina o **Cliente**.

Anexo A: Scripts de suporte ao trabalho

Scripts **fornecidos,** com Mensagens de **sucesso**, **erro, debug** e **validação de programas**:

Mensagens de output com Erro (com exemplos): Macro `so_error(<passo>, <Mensagem>, [...])`

A sintaxe dos argumentos "`<Mensagem>, [...]`" é semelhante à de `printf()`; esta macro, tem como output no STDOUT:
"`@ERROR {<Passo>} [<Mensagem>, [...]]`"

Exemplos de invocação:

- Em **S1**, O ficheiro **bd_utilizadores.dat** não existe:
  - `so_error("S1", "");`
- Em **C12**, O **Cliente** deu Timeout:
  - `so_error("C13", "Timeout Cliente");`

Mensagens de output com Sucesso (com exemplos): Macro `so_success(<passo>, <Mensagem>, [...])`

A sintaxe dos argumentos "`<Mensagem>, [...]`" é semelhante à de `printf()`; esta macro, tem como output no STDOUT:
"`@SUCCESS {<Passo>} [<Mensagem>, [...]]`"

Exemplos de invocação:

- Em **S4**, o **Servidor** armou corretamente os sinais SIGINT e SIGCHLD:
  - `so_success("S3", "");`
- Em **C5**, o **Cliente** indica que iniciou o período de espera:
  - `so_success("C6", "Espera resposta em %d segundos", MAX_ESPERA);`
- Em **S5**, o **Servidor** leu um pedido do **Cliente** na forma de um elemento **Login**:
  - `so_success("S4", "%d %s %d", request.nif, request.senha, request.pidCliente);`

Mensagens de Debug: Apesar de não ser necessário, disponibilizou-se também uma macro para as mensagens de debug dos scripts, dado que será muito útil aos alunos: **Macro `so_debug(<Mensagem>, [...])`**

A sintaxe dos argumentos "`<Mensagem>, [...]`" é semelhante à de `printf()`; esta macro, tem como output no STDOUT:
"`@DEBUG:<Source file>:<line>:<function>: [<Mensagem>, [...]]`"

Exemplos de invocação:

- Em **C6**, simplesmente para indicar um teste de passagem por uma parte do código:
  - `so_debug("Passei por aqui");`

Tem a vantagem de que mostra sempre as mensagens de debug (não precisa sequer ser nunca apagado). Quando os alunos quiserem apagar as mensagens de debug, basta descomentar a seguinte linha do programa atual:

```
//#define SO_HIDE_DEBUG       // Uncomment this line to hide all @DEBUG statements
```

E, assim, não precisam de apagar as invocações à macro so_debug, mantendo os vossos programas intocados.

Manipulação de ficheiros binários:

Neste trabalho são armazenadas informações em ficheiros em formato binário, **bd_utilizadores.dat** e **bd_produtos.dat**. Não é fácil visualizar este ficheiro usando a aplicação **cat**. Uma das formas sugeridas de analisar estes ficheiros é usando as aplicações **hexdump** ou **xxd**. No entanto, para facilitar esta tarefa, foi fornecido um script que ajuda a visualizar os conteúdos deste ficheiro (ver KB Moodle correspondente), que estão de acordo com a estrutura **Login** e **Produto**:

```
typedef struct {
    int nif;                    // Número de contribuinte do utilizador
    char senha[20];             // Senha do utilizador
    char nome[50];              // Nome do utilizador
    int saldo;                  // Saldo do utilizador
    int pid_cliente;            // PID do processo Cliente
    int pid_servidor_dedicado;  // PID do processo Servidor Dedicado
} Login;
```

Assim, o comando:

```
$ ./utils/so_show-binary-login.sh bd_utilizadores.dat
```

Mostra o resultado:

```
> ./utils/so_show-binary-login.sh bd_utilizadores.dat
| 235123532 | qwerty | Paulo Pereira | 123 | -1 | -1 |
| 234580880 | 12qwaszx | Catarina Cruz | 50 | -1 | -1 |
| 215654377 | 09polkmn | Joao Baptista Goncalves | 20 | -1 | -1 |
```

O comando **./utils/so_show-binary-produto.sh** mostra o mesmo, mas para o ficheiro de produtos.

Da mesma forma, foram desenvolvidas duas ferramentas utilitárias que leem informações de um ficheiro de texto, sendo que nesse ficheiro de texto, cada linha de texto corresponde a um registo, em que cada um dos campos desse registo são separados por um caracter separador, e produz como output um ficheiro binário de elementos do formato **Login** (ou **Produto**) acima descrita.

Assim, o comando:

```
$ ./utils/so_generate-binary-login.exe utilizadores.txt : bd_utilizadores.dat
```

Lê os utilizadores que estão no ficheiro utilizadores.txt:

```
235123532:qwerty:Paulo Pereira:123:-1:-1
234580880:12qwaszx:Catarina Cruz:50:-1:-1
215654377:09polkmn:Joao Baptista Goncalves:20:-1:-1
```

sendo que cada registo de utilizador, como se pode ver acima, ocupa uma linha de texto, e os vários campos do registo estão separados pelo caracter ':', e produz como output o ficheiro **bd_utilizadores.dat** (que é o mesmo ficheiro que foi fornecido aos alunos). O ficheiro **utilizadores.txt** pode ser encontrado na diretoria "utils". De forma análoga, foi criada a aplicação **so_generate-binary-produto.exe**, que faz o mesmo, mas para um ficheiro de produtos.

Poderá também editar este ficheiro de texto para produzir outro de formato similar, que tenha apenas um utilizador, e assim produzir um ficheiro binário com apenas um elemento **Login**, por exemplo, o ficheiro **pedido-cliente.txt**:

```
234580880:12qwaszx:sem_nome:0:123456:-1
```

Este ficheiro de texto define um utilizador com os dados (NIF e Senha) de um utilizador existente na BD (Catarina Cruz), e com um pidCliente de valor fictício de 123456. Se agora usarmos esta ferramenta desenvolvida:

```
$ ./utils/so_generate-binary-login.exe pedido-cliente.txt : pedido-cliente1.dat
```

Script Validador do trabalho:

Como anunciado nas aulas, está disponível para os alunos um script de validação dos trabalhos, para os alunos terem uma noção dos critérios de avaliação utilizados.

Passos para realizar a validação do vosso trabalho:

- Garantam que o vosso trabalho (i.e., os ficheiros `*.c *.h`) está localizado numa diretoria local da vossa área. Para os efeitos de exemplo para esta demonstração, assumiremos que essa diretoria terá o nome **`parte-3`** (mas poderá ser outra qualquer);
- Posicionem-se nessa diretoria **`parte-3`** da vossa área:

  ```
  $ cd parte-3
  ```
- Deem o comando $ **`pwd`** , e validem que estão mesmo na diretoria correta;
- Deem o comando $ **`ls -l`** , e confirmem que todos os ficheiros **`*.c *.h`** do vosso trabalho estão mesmo nessa diretoria, e também está a diretoria do validador **`so-2022-trab3-validator`**;
- Agora, posicionem-se na subdiretoria do validador:

  ```
  $ cd so-2022-trab3-validator/
  ```
- E, finalmente, dentro dessa diretoria, executem o script de validação do vosso trabalho, que está na diretoria "pai" (**`..`**)

  ```
  $ ./so-2022-trab3-validator.py ..
  ```
- Resta agora verificarem quais dos vossos testes "passam" (✓) e quais "chumbam" (✗);
- Façam as alterações para correção dos vossos scripts;
- Sempre que quiserem voltar a fazer nova validação, basta novamente posicionarem-se na subdiretoria **`so-2022-trab3-validator`** e correrem o script de validação como demonstrado acima;
- A aplicação **`so-2022-trab3-validator.py`** tem algumas opções que podem ser úteis aos alunos:
  - Se fizerem **`so-2022-trab3-validator.py -h`**, (ou **`--help`**), podem ver as várias opções;
  - Se usarem a opção **`-d`** (ou **`--debug`**), podem visualizar as mensagens que colocaram no código usando **`so_debug`**, **`so_success`**, e **`so_error`**. Caso contrário essas mensagens serão omitidas;
  - Se usarem a opção **`-e`** (ou **`--stoponerror`**), o validador irá parar assim que encontrar o primeiro erro, o que pode ser prático para não terem um output muito extenso;
  - Se usarem a opção **`-s`** (ou **`--server`**), o validador apenas irá validar o servidor (ou seja, não irá validar o cliente); se usarem a opção **`-c`** (ou **`--client`**), o validador apenas irá validar o cliente (ou seja, não irá validar o servidor); se usarem as duas, nada será validado.